SVG Web Environment for Z Specification Language

Jing Sun[1], Hai Wang[2], Sasanka Athauda[1], and Tazkiya Sheik[1]

[1] Department of Computer Science,
The University of Auckland, New Zealand
`j.sun@cs.auckland.ac.nz`
`{sath002, fshe009}@ec.auckland.ac.nz`
[2] Department of Computer Science,
The University of Manchester, United Kingdom
`hai.wang@cs.man.ac.uk`

Abstract. This paper presents a web environment for the Z formal specification language using the Scalable Vector Graphics (SVG) technology. The Z Specification Web Editor (ZSWE) is the first prototype of a web based graphical editor for the Z specification language. It not only supports graphical editing and global accessibility for the Z formal specifications, but also provides model comprehension facilities such as schema expansion, specification navigation and model querying. This paper outlines the requirement, design and implementation of the tool and its future improvements.

Keywords: Z formal specification language, Web based tool support, Scalable Vector Graphics.

1 Introduction

Formal methods is defined as mathematically based techniques for the specification, development and verification of software and hardware systems [1]. The well-defined semantics and syntax of formal specification languages make them suitable for precisely capturing and formally verifying system requirements. Z is a formal specification language based on set theory and predicate logic [2]. It has been widely used in both industry and academic research for the specification and verification of software systems. The World Wide Web (WWW) acts as a promising environment for software specification and design because it allows sharing design models and providing hyper textual links among the models. Formal methods like the CafeOBJ system [3] have included an environment supporting formal specification over the Internet. Schemas using pure Z notation on the web based on HTML and Java Applet have also been investigated by Bowen et al. [4] and Ciancarini et al. [5]. Although HTML has been successful in presenting information on the Internet, the lack of content information and the overburdened use of the display tags have made the efficient retrieval and exchange of information content more difficult to achieve. In 2001, Sun et al. proposed an XML/XSL approach in presenting the Z/Object-Z languages on the web [6]. It uses the XSL Transformation language to translate the XML form of Z/Object-Z models into HTML for automated browser display. In their approach, XML has been introduced as an interchange format for documenting Z specifications. However, the graphical support of the resulting Z model display was still restricted on using HTML only. In this paper, we present an approach of using the Scalable Vector Graphics (SVG) to implement a web based environment for the Z specification language[1]. SVG is a World Wide Web Consortium (W3C) recommended language for describing two-dimensional graphics and graphical applications in XML. It can overcome the poor graphical support in using HTML for displaying Z specifications on the web. The Z Specification Web Editor (ZSWE) prototype tool presented in the paper uses the standard Z Markup Language (ZML) format defined by Utting et al. in 2003 [7]. The ZSWE tool not only supports true graphical editing for the Z formal specifications, but also provides model comprehension facilities such as schema expansion, specification navigation and model querying. There is some related work in providing editing facility for the Z notation, such as the functions in Z/EVES, ZETA and CADiZ. From an editing support point of view, most of those tools only provide limited editing facility of Z specifications. Compared to our approach, they are lack of additional model comprehension functions such as specification navigation and model querying. In comparison with other approaches in presenting Z models on the web, our SVG prototype tool also provides a better graphical display and editing supports for Z models over the internet.

The remainder of the paper is organized as follows. Section 2 introduces background information on Z, ZML and SVG. In section 3, we discuss the various aspects regarding a specification environment for the Z language. Section 4 presents the architecture design of the web based specification prototype tool - ZSWE. In section 5, we present some implementation issues of the ZSWE. Section 6 gives an overview of the prototype tool. Section 7 concludes the paper and discusses future improvements.

2 Background

2.1 The Z Formal Specification Language

Z [2] is a state-based formal language based on ZF set theory and first-order predicate logic. It is specially suited to model system data and state changes. A Z specification typically includes a number of state and operation schema definitions. A state schema encapsulates variable declarations and related invariant predicates. An operation schema defines the relationship between the 'before' and 'after' states corresponding to one or more state schemas. Complex schema definitions can be composed from the simple ones using the schema calculus. Z has been widely adopted to specify a range of software systems. The following is a state schema example of a Birthday Book specification taking from [2].

[1] This work was supported in part by HyOntUse Project (GR/S44686) funded by the UK Engineering and Physical Science Research Council.

$$\begin{array}{|l}
\multicolumn{1}{l}{\textit{BirthdayBook}} \\
\textit{known} : \mathbb{P}\,\textit{NAME} \\
\textit{birthday} : \textit{NAME} \mathrel{\ooalign{\hfil$\mapstochar\mkern5mu$\hfil\cr$\to$\cr}} \textit{DATE} \\
\hline
\textit{known} = \operatorname{dom} \textit{birthday} \\
\hline
\end{array}$$

The above defines a basic structure of a birthday book. The variable `known` represents the set of people in the birthday book; and the variable `birthday` is a partial function that associates the people's names with their birth dates. The state invariant imposes that the `known` set is set of the people who already had their birthday recorded. Other operations such as 'add' or 'find' a birthday record can be defined accordingly. In this paper, we will be using this Birthday Book example to illustrate the requirements of a Z specification editor.

2.2 The Z Markup Language

EXtensible Markup Language (XML) is a global standard for representing information in a textual format. The Z Markup Language (ZML) is defined to serve as an XML interchange format for documenting Z specifications by Utting et al. in [7]. Its syntax was based on the Z ISO International Standard format [8]. It is recommended by the Community Z Tools Initiative (CZT) group that future tool development on Z should follow this XML convention. In addition, a library of Java classes has been developed for the parser support of the ZML files. The following denotes a partial ZML representation of the variable declaration '$\textit{known} : \mathbb{P}\,\textit{Name}$' in the Birthday Book state schema example.

```
<VarDecl>
  <DeclName>
    <Word>known</Word>
  </DeclName>
  <PowerExpr>
    <RefExpr Mixfix="false">
      <RefName>
        <Word>NAME</Word>
      </RefName>
    </RefExpr>
  </PowerExpr>
</VarDecl>
```

We can see from the above example that ZML has a complex syntax structure and it is intended for machine (tool) interpretation only.

2.3 Scalable Vector Graphics

Scalable Vector Graphics (SVG) is a language for describing two-dimensional graphics on the web using a standard XML format [9]. It supports three types of graphic objects:

– **Vector graphic shape**: SVG provides pre-defined graphical shapes and a path element which can be used to create any arbitrary two-dimensional shape.
– **Text**: SVG has several elements that displays text in different layouts.
– **Image**: SVG supports other types of graphical images to be embedded in SVG documents.

The data representation of conventional images is quite different to SVG. Conventional images are broken into small pixels and the description (e.g., color of the pixel) of each of these pixels has to be stored. Therefore, these images hold a large file size. On the other hand, SVG provides the type of shape required to be drawn, the coordinates, and the style of the shape in XML format. This information can be translated by the SVG plug-in on the web browser as the shape is displayed. An example of an SVG file that generates a simple rectangle is shown below:

```
<svg width="100" height="100">
  <rect x="10" y="10" width="50" height="50" style="fill:red"/>
</svg>
```

As shown in this example, the 'rect' tag informs the browser that the shape is a rectangle with the coordinates and style of the rectangle provided. SVG also supports the following aspects, which we found useful in developing the ZSWE prototype tool:

– **Animation support**: SVG provides animation support on graphical shapes. Such animation support includes dynamically changing the location, size, style of a shape.
– **Zoom-in and zoom-out**: SVG supports zoom in/out features on its graphical shapes. The graphical quality of the shape is maintained during the zoom-in and zoom-out.
– **Unicode support**: SVG provides support to display Unicode symbols.
– **DOM functionality**: Since SVG is in XML format, other programming languages can use the DOM functions to create the SVG DOM which can be used to locate and access SVG content information.

3 Aspects of a Z Specification Editor

In this section we describe some of the key issues related to a web based editor for the Z specification language. We summarize our requirements into five different aspects, i.e., graphical display, schema expansion, specification navigation, model querying, and specification validation.

3.1 Graphical Display

A Z specification consists of schema boxes and mathematical expressions. Z is a language based on set theory and predicate logic, which consists of a rich set of mathematical symbols. The following defines the *AddBirthday* operation schema in the Birthday Book example.

AddBirthday

$\Delta BirthdayBook$
$name? : NAME$
$date? : DATE$

$name? \notin known$
$birthday' = birthday \cup \{name? \mapsto date?\}$

The `AddBirthday` operation allows the users to add new birthday records into the system based on the pre-condition that the person has not been recorded before. From the above example, we can see that the first requirement of a Z specification editor is to support elegant graphical display of Z schema box drawings and the usage of mathematical symbols such as '$\notin$','$\cup$, '$\mapsto$' and so on.

3.2 Schema Expansion

In a Z specification, the full definition of a schema can be obtained by expanding the inclusion section of the schema. For example, the expanded view of the *AddBirthday* schema in the previous subsection is as follows.

AddBirthday

$known, known' : \mathbb{P}\, NAME$
$birthday, birthday' : NAME \nrightarrow DATE$
$name? : NAME$
$date? : DATE$

$known = \mathrm{dom}\, birthday$
$known' = \mathrm{dom}\, birthday'$
$name? \notin known$
$birthday' = birthday \cup \{name? \mapsto date?\}$

The above gives the full definition of the `AddBirthday` schema by expanding the definitions inside the '*ΔBirthdayBook*' expression. Another form of expansion is the Z schema calculus. In a Z specification, complex operations can be constructed by using schema calculus operators such as '$\wedge$', '$\vee$' and so on. For example, a 'robust' version of the `RAddBirthday` operations can be specified by using the conjunction and disjunction operators on the `AddBirthday`, `Success` and `AlreadyKnown` schemas in the Birthday Book example as follows.

Success

$result! : REPORT$

$result! = ok$

AlreadyKnown

$\Xi BirthdayBook$
$name? : NAME$
$result! : REPORT$

$name? \in known$
$result! = already\_known$

$$RAddBirthday \mathrel{\widehat=} (AddBirthday \land Success) \lor AlreadyKnown$$

The `RAddBirthday` operation will insert a new record into the birthday book or report the record has already been stored. The full definition of the `RAddBirthday` schema can be obtained by expanding the definitions in the `AddBirthday`, `Success` and `AlreadyKnown` schemas as follows.

$$\begin{array}{|l}
\multicolumn{1}{l}{RAddBirthday}\\
\hline
known, known' : \mathbb{P}\, NAME \\
birthday, birthday' : NAME \pfun DATE \\
name? : NAME \\
date? : DATE \\
result! : REPORT \\
\hline
known = \operatorname{dom} birthday \\
known' = \operatorname{dom} birthday' \\
(name? \notin known \land birthday' = birthday \cup \{name? \mapsto date?\} \land result! = ok) \\
\lor (name? \in known \land birthday' = birthday \land result! = already\_known) \\
\hline
\end{array}$$

Other forms of schema operators include schema composition '$\fatsemi$', implication '$\Rightarrow$', negation '$\neg$' and piping '$\gg$', which have been discussed in many Z books [2]. The schema expansion is useful for analysis, review and reasoning about Z specifications. For instance, in the case of calculating the pre-/post-conditions related to a particular scheme operation, it is necessary to expand (unfold) the full definition of schema before the calculation. Thus, the second requirement of the Z specification editor is to support automatic schema expansions to display a full definition of a schema as needed.

3.3 Specification Navigation

In a large Z model that contains quite a number of schemas, it is sometimes hard for the users to keep track of all the definitions. In this case, it is desirable for the users to be able to navigate from one point of definition to another by referring to a schema name or a variable type. For example, in the following `FindBirthday` schema, if the users would like to refer to the original definition of the `BirthdayBook`, they should be able to navigate to its point of definition by referring to the `BirthdayBook` schema name inside the operation. Similarly, variable types, such as `NAME`, `DATE`, etc., should have the navigation facility as well.

$$\begin{array}{|l}
\multicolumn{1}{l}{FindBirthday}\\
\hline
\Xi BirthdayBook \\
name? : NAME \\
date! : DATE \\
\hline
name? \in known \\
date! = birthday(name?) \\
\hline
\end{array}$$

As mentioned earlier, this kind of navigation feature is very useful when the user is dealing with a large Z specification that contains quite a number of schema definitions. It will not only help the user to obtain a good understanding of the relationships among the schemas, but also provide easy accessibility for all the definitions in the specification. Thus, the third requirement of a Z specification editor is to support specification navigation that allows the users to navigate from one point of the definition to another inside a Z model.

3.4 Model Querying

The idea of querying a Z model comes from the concept of specification comprehension, i.e., to obtain a better understanding of what has been modeled in the specification. In general, specification comprehension is analogous to program understanding. But the former is more complicated than program understanding because programs are executable, while specifications are not necessarily to be so [10]. Thus it is desirable for a specification tool to provide means for the users to enhance the understanding of the static properties of a formal model it represents. The query of a Z model is to fulfill such a comprehension facility. We summarized four types of query functions on a Z model as follows.

– **Schema query**: provides information on the schemas in a Z model, such as where this schema is used and how it is used, i.e., being included in or modified in other schemas.
– **Variable query**: provides information on the variables in a schema, such as the type of variables (state/input/output), in which schema or operation the variable is defined or used, etc.
– **Operation query**: provides information on the operations in a Z model, such as the variables and predicates that an operation has and so on.
– **Reference query**: provides cross-reference information on the schemas in a Z model.

Querying is considered as a usability function. It is not an implicit element of a Z specification. For example, Z schemas, functions, and its variables and predicates can be considered as containing implicit elements of the Z model. But model querying is search functionality for locating these implicit elements and providing a better understanding of the underlying Z specification. Thus, the fourth requirement of the Z specification editor is to support model querying functions that allows the users to explore the static properties inside a Z model.

3.5 Specification Validation

Specification validation denotes the process of determining whether the specification is correct and a true reflection of the requirements that is meant to capture. We summarized three levels of validation associated to a particular Z model.

- **Syntax checking**: to check whether a Z expression is written properly according the Z language syntax.
- **Type checking**: to check whether an expression is correct according to the type checking rules of the Z language. For example, we could define a syntactically correct expression such as '$\mathtt{x} : \mathbb{N}_1$' where '`x`' takes values from the positive nature number set. But if we later assign a value of negative integer to '`x`', this is where a type inference error is occurred. Type checking [11] techniques are usually applied for validating these kind of errors in a specification.
- **Semantic checking**: to check the logical correctness of a Z specification. Even if a Z specification is syntactically and type correct, there are still possibilities that the logical statements in the model might conflict each other or the dynamic behaviors of the model does not truly reflects the requirement. These errors are related to the semantic meanings of the Z model. Semantic checking usually requires more complicated techniques than that of syntax and type checking. In general, theorem proving and specification animation [12] are two approaches that can be used for the semantic checking process.

We believe that a Z specification editing tool should provide some mechanism to allow the users to validate whether their specifications are correct. Thus, our last requirement of a Z specification editor is to support specification validation for checking the correctness of a Z model. In this section, we discussed some general aspects related to a Z specification editing tool. And our prototype implementation should closely follow some of these requirements.

4 Architecture Design of the ZSWE

Software architecture is an important level of description for the development of software systems. It represents the high level structure of a system, which comprise the definitions of software components involved, the external visible properties of those components, and the communications among the components. When consider the architecture of a web based application, there are two major type of approaches, i.e., client-side based architecture and server-side based architecture. Client-side applications are loaded from server and reside in memory of the client machines. Complicated computations are done on the client machines without having to request them from the server. Although this lessens the overburdens on the server, the initial loading and response times of client-side applications are slow. This is one of the major drawbacks of the approach. On the other hand, server-side architecture handles all the complex computations on the server and sends the results back to the client. Therefore, the client machine is not under heavy load and acts as a simple web browser that posts requests and display the results. This also provides the use of less bandwidth and gaining faster web responses, as all processing is done on the server and web pages are dynamically presented. In nowadays, as the web servers become increasingly powerful in terms of the computational capability, more and more web based softwares choose the server-side architecture to provide an easy accessible and 'thin client' application. We decided to implement our ZSWE prototype tool on the server side for the same reason. Figure 1 shows an overview of the sever-side component architecture of the tool.

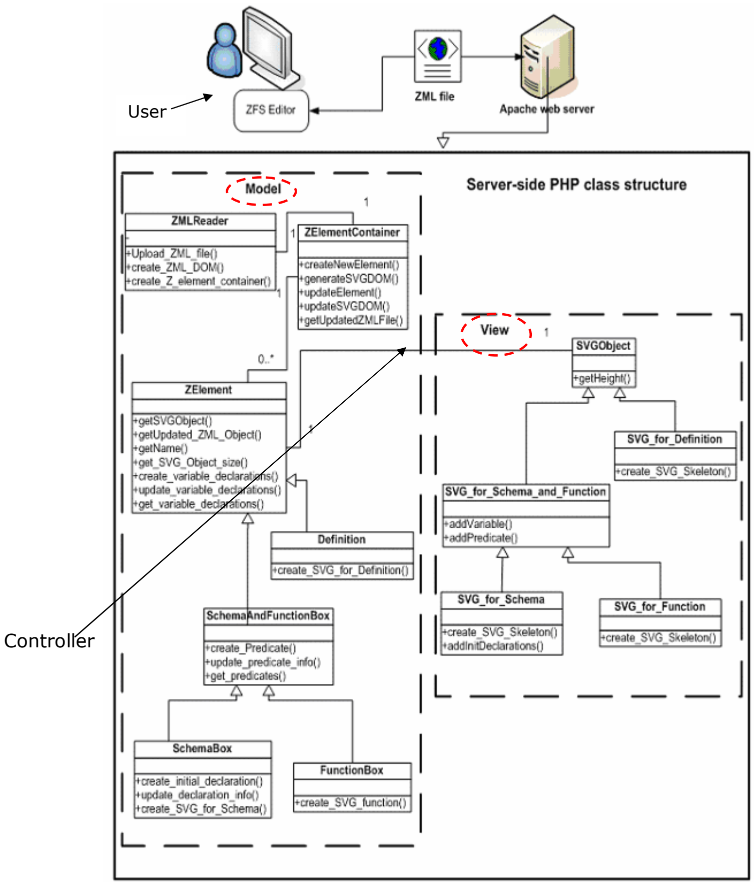


Fig. 1. ZSWE server-side architecture

As shown in the diagram, the client-side consists of the web browser, SVG plug-in, and the client requests. It is the communication point between the server and the actual client. The user can create/upload a Z specification to the server, modify the information in the Z specification, or download an updated Z specification from the server. Note that the standard ZML syntax mentioned in section 2.2 is chosen as the input/output interchange format for documenting the Z specification in our tool. The server-side of the tool consists of components that handles the corresponding computation of SVG elements. When a ZML file is uploaded to the server, the '`ZMLReader`' component on the server processes the file and generates the ZML-DOM representation of the specification. This DOM is then passed to the '`ZElementContainer`' component which creates the '`ZElement`' objects according the information presented in the ZML-DOM. The users can add new Z definitions, or update the existing definitions of a Z model. This is also performed by sending the updating information to the server, the '`ZElementUpdater`' component on the server finds the corresponding '`ZElement`' object from the '`ZElementContainer`' and performs the update. Once each of the '`ZElement`' objects has been created or updated, a SVG representation of the specification is generated through the '`SVGGenerator`' component. After the server creates the graphical representation of the Z model in SVG format, it is sent back to the client as a SVG file. The SVG plug-in which runs inside the web browser identifies this file as a SVG document, and translates its tags into proper graphical elements. Finally, these graphical elements are displayed in the web browser.

Figure 1 also describes the architecture in a Model View Controller (MVC) structure. MVC is a common architecture used by the modern software developers to increase modularity of the code. It divides the code into three modules: Model, View, and Controllers, which enables data flow between the Model and View via the Controller. Each of these three modules acts independently to maintain consistency. MVC architecture is commonly used in server-side development because it enables the maintenance of multiple views of the same system. As highlighted in the above diagram, the class structures is divided in to major sub-components: Model, View and Controller. The Model contains the components such as '`ZMLReader`', '`ZElementContainer`', '`ZElement`' and so on, to read the ZML file, creates its DOM, breaks the ZML-DOM into Z elements such as schemas, functions, definitions etc. The View can be considered as all the SVG content related components such as the '`SVGObject`', and the server-side PHP script that generates them because these SVG files contain the graphical interface of the tool. Finally, the data updating components such as the '`ZElementUpdater`' and the PHP script that maps the user actions to model updates can be considered as the Controller elements of the MVC model. And the updater script that performs the updates of SVG representation and the Z element object also act as a Controller element.

5 Implementation Issues of the ZSWE

The main techniques involved in the implementation of the ZSWE prototype tool are Scalable Vector Graphics (SVG), Hypertext Pre-processor (PHP) and ECMA scripting.

5.1 SVG and ECMA Scripting

Our main issue in the implementation was to combine data information between SVG and HTML. Since both HTML and SVG support web scripting functions, ECMA scripting is used for the implementation. ECMA scripting is a standard for describing a web scripting language that can create a rich environment for a web site. It provides built-in methods and classes to support XML-DOM. A web browser allows ECMA scripting for client-side computing and also provides for events such as mouse events, change of focus, image and page loading, selection and form submission etc. It can be embedded in HTML and PHP to allow for animation of objects and events. This scripting language is very useful in web application as it provides the functionalities of object oriented programming that cannot be achieved by using plain HTML. The following is an example that uses ECMA scripting to perform dynamic updates on SVG elements by catching the events triggered by these elements:

```
<svg width="100" height="100" onload=init(evt)>
 <desc>
  <script language="text/ecmascript">
   <![CDATA[
   var svgdoc;
   function init(evt){
    svgdoc=evt.getTarget().getOwnerDocument();
   }
   function mousePress(name){
    var element=svgdoc.getElementById(name);
    element.setAttribute ("height", "40");
   }
   ]]>
  </script>
  <rect id="rectangle" x="10" y="10" width="40" height="100"
   style="fill:red" onmouseclick="mousePress('rectangle')"/>
 </desc>
</svg>
```

As show in the above code segment, the ECMA functions are embedded inside the SVG content. When these SVG content is loaded onto a web browser, the '`onload`' event calls the '`init`' method in the ECMA script. This method assigns the SVG-DOM root object to a variable. As highlighted above, the SVG '`rect`' element has an unique ID and a mouse-click event. When the user clicks on the rectangle, the '`onmouseclick`' event calls the '`mousePress`' method in the ECMA script, and sends the rectangle ID as its input. This method uses the ID to get the rectangle object from the SVG-DOM. Then it changes the height attribute of the rectangle object, which dynamically effects on the graphical display.

5.2 SVG and PHP

When developing web based applications on the server-side, there are special programming languages to handle the computation mechanism. We chose the Hypertext Pre-processor (PHP) script language in our implementation. PHP provides a good set of functions that are used for extracting and modifying information from XML documents. PHP version 4 and above includes functions that can generate XML documents using XML-DOM. Therefore, the SVG code can be generated by creating the XML nodes that represent the SVG pages. The following is a segment of PHP code for generating the SVG example shown in section 2.3.

```
$root = $doc->create_element("svg");
$root = $doc->append_child($root);
$root->set_attribute ("width", "100");
$root->set_attribute ("height", "100");
$rect = $doc->create_element("rect");
$rect = $root->append_child($rect);
$rect->set_attribute ("x", "10");
$rect->set_attribute ("y", "10");
$rect->set_attribute ("width", "50");
$rect->set_attribute ("height", "50");
$rect->set_attribute ("style", "fill:read");
```

The code segment displayed above first creates a new SVG canvas and set its size, then create a SVG rectangle element and appends it to SVG canvas as a child node. The position, size and the style attributes of the SVG rectangle are set accordingly. From the above example, we can see that by using PHP we are able to easily generate and modify SVG representations of Z models and sent them back to the web browser for displaying.

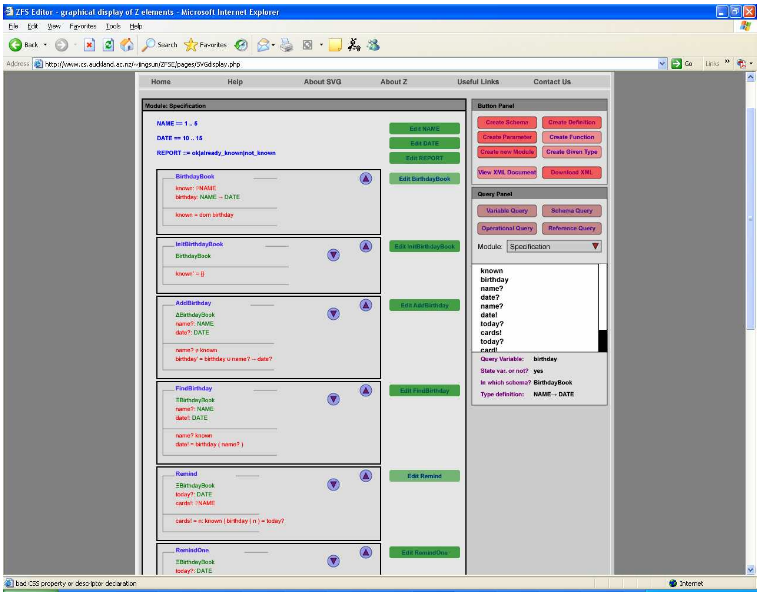
6 ZSWE in Action

The ZSWE prototype tool consists of three main pages, i.e., the index page, the SVG display page and the Z schema editing page[2].

6.1 SVG Z Model Display

The SVG display page provides the main functionalities of the tool, such as model display, schema expansion, navigation and querying. Figure 2 illustrates the SVG display page of the ZSWE tool. The Z specification model is displayed on the left hand side of the page. Navigation points are provided for each of the schema names and type declarations to allow quick references among the definitions. The down arrows represent expansion points inside the specification to allow the user to view the full definition of a schema. The schema expansion function was implemented using the SVG animation technique. In addition, a zoom in/out feature was provided for the display page to allow the user to zoom in and out on the Z models. On the right hand side of the page are the button panel and the query panel. The button panel contains buttons for creating new schema, give type, axiom definition and so on. There are two additional buttons

[2] The Z Specification Web Editor (ZSWE) prototype tool is available at `http://www.cs.auckland.ac.nz/~jingsun/ZFSE/pages/index.php`



Module: Specification

NAME == 1 .. 5
DATE == 10 .. 15
REPORT ::= ok|already_known|not_known

BirthdayBook
known: ℙNAME
birthday: NAME → DATE
known = dom birthday

InitBirthdayBook
BirthdayBook
known' = {}

AddBirthday
ΔBirthdayBook
name?: NAME
date?: DATE
name? ∉ known
birthday' = birthday ∪ name? ↦ date?

FindBirthday
ΞBirthdayBook
name?: NAME
date!: DATE
name? known
date! = birthday (name?)

Remind
ΞBirthdayBook
today?: DATE
cards!: ℙNAME
cards! = n: known | birthday (n) = today?

RemindOne
ΞBirthdayBook
today?: DATE

Module: Specification

known
birthday
name?
date?
name?
date!
today?
cards!
today?
card!

Query Variable: birthday
State var. or not? yes
In which schema? BirthdayBook
Type definition: NAME→ DATE

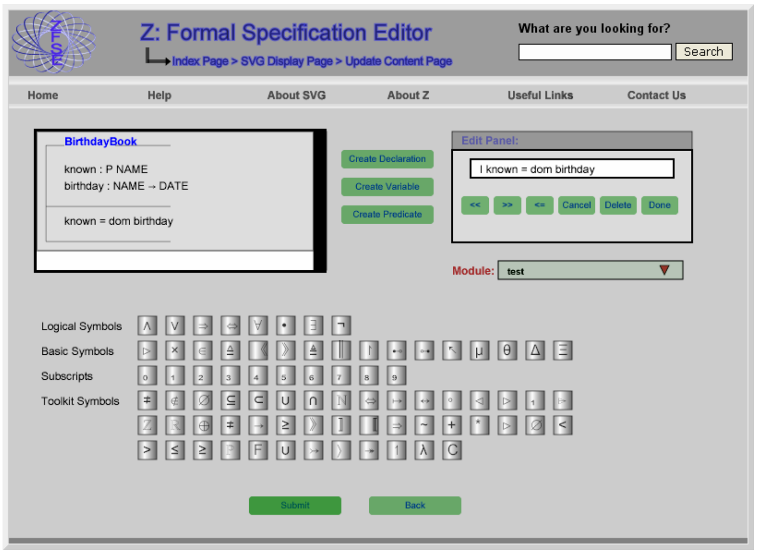
Fig. 2. The SVG display page

on bottom of the button panel where one is used to view the ZML document of the current Z specification, and the other to download the ZML file.

Model querying is an important functionality that has been implemented in the ZSWE tool. The query panel consists of four types of querying buttons, i.e., variable query, schema query, operational query, and reference query. Each of the query functions provides model comprehension facilities described in section 3.4. For example, in the case of a schema query, when the user click on the '`Schema Query`' button, a list of all the schema names in the Z model are displayed on the query panel. After one of these listed schema names is selected, the query panel provides the names of other schemas that has extended or used the selected schema.

6.2 Z Schema Editing

By clicking on the editing the Z definitions button in Figure 2, a Z schema editing page can be invoked to allows the user to make changes to the schema definitions. The schema editing page can also be opened by the create new definition buttons on the SVG display page. Figure 3 shows the functionality provide by the ZWSE schema editing page. New variables and predication definitions of a schema are input on the right panel and updated to the SVG display panel on the left. A mathematics symbol panel is provided to assist the user for inputting Z mathematical expressions. It uses the SVG Unicode representations for the displaying of the mathematical symbols. Note that our mathematics symbol panel



BirthdayBook

known : P NAME
birthday : NAME → DATE

known = dom birthday

Edit Panel: known = dom birthday

Module: test

Fig. 3. Z schema editing page

adopts some of the symbol layout from the Z/EVES tool. After the modifications have been made, new updates on the specifications are displayed on the main SVG display page. The user can download an updated version of the ZML file of the Z specification. As we mentioned in section 3.5, a Z specification editor should provide some mechanism for checking the correctness of the Z model. In our ZSWE prototype tool, we have implemented a syntax checking facility for validating whether the Z specification is written according to a proper Z syntax. The syntax checking is based on XML schema validation mechanism. Every time when a Z specification is uploaded, its XML representation is validated against the ZML schema definition.

7 Conclusion

In this paper, we presented a web environment for the Z formal specification language. Different aspects of a Z specification editor were discussed. The design and implementation of the Z Specification Web Editor (ZSWE) prototype tool using the SVG technology was presented. Our ZSWE tool not only supports graphical editing and global accessibility for the Z formal specifications on the internet, but also provides model comprehension facilities such as schema expansion, specification navigation and model querying. In addition, the ZSWE tool also provides a basic Z syntax checking facility.

For the future extensions, firstly, our idea of the Z web environment can be easily adopted to other formal specification languages such as Object-Z [13], TCOZ [14] and so on. Both Object-Z and TCOZ have XML representations of their langauge syntaxes, thus such extensions should be straightforward. Secondly, as our ZSWE tool only supports syntax checking facility of the Z language at the moment, one of the immediate future work could be to add type and some semantic checking facilities into the prototype environment. Finally, our Z web environment is currently an anonymous web user application. By providing a login name and password for each user, online saving of the Z specification models can be achieved. This would enable different users to work on a same Z specification model collaboratively and continuously.

References

1. From Wikipedia: (The Free Encyclopedia) Available at: `http://en.wikipedia.org/wiki/Formal_methods`.
2. Spivey, J.: The Z Notation: A Reference Manual. 2nd edn. International Series in Computer Science. Prentice-Hall (1992)
3. Futatsugi, K., Nakagawa, A.: An Overview of CAFE Specification Environment. In Hinchey, M., Liu, S., eds.: the IEEE International Conference on Formal Engineering Methods (ICFEM'97), Hiroshima, Japan, IEEE Computer Society Press (1997)
4. Bowen, J.P., Chippington, D.: Z on the Web using Java. [15] 66–80
5. Ciancarini, P., Mascolo, C., Vitali, F.: Visualizing Z notation in HTML documents. [15] 81–95
6. Sun, J., Dong, J.S., Liu, J., Wang, H.: Object-Z Web Environment and Projections to UML. In: WWW-10: 10th International World Wide Web Conference, ACM Press (2001) 725–734
7. Utting, M., Toyn, I., Sun, J., Martin, A., Dong, J.S., Daley, N., Currie, D.: ZML: XML Support for Standard Z. In: 3nd International Conference of Z and B Users (ZB'03). LNCS, Springer (2003)
8. Developed by members of the Z Standards Panel, Project Editor: Toyn, I.: Z Notation: Final Committee Draft, CD 13568.2 (1999) Available at: `http://www.cs.york.ac.uk/~ian/zstan/`.
9. World Wide Web Consortium (W3C): (Scalable Vector Graphics (SVG)) Available at: `http://www.w3.org/Graphics/SVG/`.
10. Hayes, I., Jones, C.: Specifications are not (necessarily) executable. Software Eng. Journal **4** (1989) 330–339
11. Dong, J.S., Li, Y.F., Sun, J., Sun, J., Wang, H.: XML-based static type checking and dynamic visualization for TCOZ. In: 4th International Conference on Formal Engineering Methods, Springer-Verlag (2002) 311–322
12. Sun, J., Dong, J.S., Liu, J., Wang, H.: A XML/XSL Approach to Visualize and Animate TCOZ. In: The 8th Asia-Pacific Software Engineering Conference (APSEC'01), IEEE Press (2001) 453–460
13. Smith, G.: The Object-Z Specification Language. Advances in Formal Methods. Kluwer Academic Publishers (2000)
14. Mahony, B., Dong, J.S.: Timed Communicating Object Z. IEEE Transactions on Software Engineering **26** (2000)
15. Bowen, J.P., Fett, A., Hinchey, M.G., eds.: ZUM'98: The Z Formal Specification Notation, 11th International Conference of Z Users, Berlin, Germany, 24–26 September 1998. Volume 1493 of Lect. Notes in Comput. Sci., Springer-Verlag (1998)